



UNIVERSIDAD DE GRANADA

TRABAJO FIN DE GRADO

INGENIERÍA INFORMÁTICA

Aprendizaje Automático para la extracción de características y detección de situaciones anómalas en multitudes

Autor

Diego Navarro Cabrera

Directores

Name of the main supervisor

Name of the second supervisor (if available)



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA Y
TELECOMUNICACIONES

—
Granada, 7 de mayo de 2021

Aprendizaje Automático para la extracción de características y detección de situaciones anómalas en multitudes

Diego Navarro Cabrera

Palabras clave:

Resumen

Machine learning for feature extraction and abnormal crowd behavior

Diego Navarro Cabrera

Keywords:

Abstract

Yo, **Diego Navarro Cabrera**, alumno del Grado de Ingeniería Informática de la **Escuela Técnica Superior de Ingeniería Informática y Telecomunicaciones de la Universidad de Granada**, con DNI 75935043Z, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

A handwritten signature in blue ink, appearing to read "Diego", enclosed within a stylized oval or swoosh shape.

Fdo: Diego Navarro Cabrera

Granada, 7 de mayo de 2021

D. Name of the main supervisor y D.^a Name of the second supervisor (if available), profesores del Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado *Aprendizaje Automático para la extracción de características y detección de situaciones anómalas en multitudes*, ha sido realizado bajo su supervisión por **Diego Navarro Cabrera**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 7 de mayo de 2021

Los directores:



Name of the main supervisor



**Name of the second supervisor
(if available)**

Agradecimientos

Índice general

Índice general	5
1 Introducción	7
2 Fundamentación Teórica	8
2.1. Aprendizaje automático	8
2.2. Máquina de soporte vectorial	8
2.3. Extracción de características	8
2.4. Autoencoders	8
2.5. Aprendizaje multi-tarea	8
3 Descripción del problema	9
3.1. Conjuntos de datos de trabajo	9
3.1.1. UMN Crowd Dataset	10
3.1.2. Violent Flows	11
3.2. Estado del arte	12
3.2.1. Modelo basado en descriptores visuales locales de nivel medio	13
3.3. Modelo propuesto	14
3.3.1. Representación de la multitud	15
3.3.2. Descriptores visuales	17
3.3.3. Vector de características y clasificador	20
4 Resultados experimentales	23
4.1. Metodología usada	23
4.1.1. UMN Crowd Dataset	23
4.1.2. Violent Flows	23
4.2. Comparación de resultados	23
4.2.1. UMN Crowd Dataset	23
4.2.2. Violent Flows	23

5 Conclusión	24
Bibliografía	25

1 Introducción

Intro

Fundamentación Teórica

- 2.1. Aprendizaje automático
- 2.2. Máquina de soporte vectorial
- 2.3. Extracción de características
- 2.4. Autoencoders
- 2.5. Aprendizaje multi-tarea

Descripción del problema

En este capítulo desarrollaremos en más profundidad el problema a tratar. Como ya hemos mencionado anteriormente, nuestro objetivo es estudiar el comportamiento de una multitud para detectar cuándo se está produciendo una anomalía.

En concreto nos centraremos en el enfoque holístico o *top-down* de este problema. Este enfoque trata a la multitud como un solo ente y analiza la dinámica del grupo más que las acciones de sus individuos. La detección de comportamientos individuales anómalos dentro de la multitud queda fuera del ámbito de este trabajo, pero algunas propuestas de las estudiadas podrían adaptarse para ello.

La estructura de esta sección será la siguiente. Primero hablaremos de los conjuntos de datos sobre los que trabajaremos. A continuación trataremos el estado del arte y explicaremos en más profundidad el modelo sobre el que nos hemos basado. Finalmente detallaremos el modelo desarrollado y diversas alternativas que se han tenido en cuenta.

3.1. Conjuntos de datos de trabajo

Se han elegido 2 conjuntos de datos ampliamente usados en el ámbito de la detección de anomalías en multitudes. El primero es el UMN Crowd Dataset [1], que consiste de una pequeña selección de vídeos en el que un grupo de personas echa a correr tras una señal. El segundo conjunto de datos es conocido como Violent Flows [2] y recoge una mezcla de vídeos de peleas y vídeos de control. Estos dos conjuntos de datos nos permitirán comprobar la efectividad del modelo ante 2 de las anomalías más habituales y relevantes en el día a día: las peleas y los escenarios de pánico. Además su amplio uso en la literatura nos permitirá comparar fielmente nuestro modelo con otros muchos. A continuación describiremos más detalladamente ambos conjuntos.

3.1.1. UMN Crowd Dataset

Este conjunto de datos, elaborado por la Universidad de Minnesota (UMN), consiste de 11 vídeos en los que un grupo de personas se comporta de manera normal hasta que recibe una señal y todos se dispersan corriendo. Los 11 vídeos se dividen en 3 escenas distintas, cada una con 2, 6 y 3 vídeos respectivamente. Todos estos vídeos se encuentran unidos en un mismo archivo que conviene dividir antes de ser procesado.



Figura 3.1: Ejemplo de fotograma normal (izqda) y anómalo (drcha) en la escena 1.

Este dataset clasifica los fotogramas normales y los anómalos por medio de un cartel que aparece en pantalla cuando se produce la anomalía, tal y como se puede ver en 3.1. Para evitar que esto pueda influir en nuestro modelo se ha recortado la parte superior de cada vídeo para eliminar dicho cartel. Por otro lado, tal y como se detalla en [3] estos carteles no son exactos, ya que aparecen significativamente después del comienzo de la anomalía. Es por esto que se ha decidido usar las anotaciones provistas en dicho artículo para etiquetar el comienzo de la anomalía en cada vídeo. Además de esto, en la mayoría de los casos la anomalía acaba antes de que empiece el siguiente vídeo, por lo que también tendremos que anotar el fotograma de finalización de cada caso. Hablaremos con más detalle de esto en el apartado de experimentación.

A pesar de sus ventajas y su uso muy extendido, este dataset presenta algunos inconvenientes que hay que tener en cuenta. Para empezar, su tamaño de 11 vídeos es bastante limitado, más aún si tenemos en cuenta que están divididos en 3 escenas que han de evaluarse de manera separada. Además solo representa anomalías de un tipo muy específico (gente echando a correr), por lo que no nos sirve para evaluar modelos enfocados a detectar todo tipo de

anomalías, algo de lo que hablaremos cuando propongamos nuestro modelo.

3.1.2. Violent Flows

Este conjunto de datos es una colección de vídeos de escenas reales para entrenar y evaluar modelos de detección de violencia. Contiene 246 instancias que se dividen en vídeos con violencia y sin ella. Se tratan de escenas cortas de entre 1 y 6 segundos y que se clasifican de manera íntegra como violentas o no.



Figura 3.2: Fotograma de ejemplo de una escena normal (izqda) y una violenta (drcha)

Los vídeos de este conjunto han sido extraídos de YouTube, y son muy distintos entre si, lo que nos permitirá comprobar la capacidad de generalización de nuestro modelo. Por otro lado, la baja resolución a la que se encuentran dificulta su análisis, ya que algunos fotogramas están tan borrosos que es difícil saber qué está pasando.

De forma similar al conjunto de datos anterior, este posee una cantidad de instancias de entrenamiento bastante limitada (246 vídeos clasificados individualmente), y se centra en un tipo de anomalías concreto, en este caso las peleas. Por otro lado, al contrario de lo que cabría esperar al trabajar con anomalías, ambas clases tienen un número de instancias muy equilibrado, 121 vídeos violentos y 125 normales, lo que nos podría facilitar el entrenamiento de un modelo de clasificación binaria, pero dificultar el uso de otras opciones centradas en entrenar únicamente con instancias normales (clasificadores de una sola clase).

3.2. Estado del arte

Existen numerosos trabajos que han probado diversas formas de detectar comportamientos anómalos. Algunas de estas son:

- Estudiar el flujo óptico de la imagen. Por ejemplo, en [4] usan el valor medio de este para estimar una interacción de fuerzas y detectar posibles zonas de conflicto, tal y como se ve en la figura 3.3.

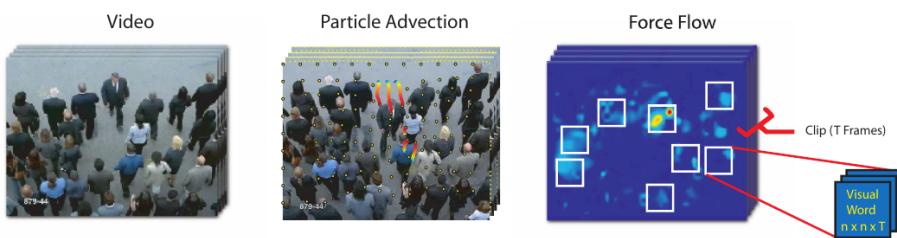


Figura 3.3: Demostración de la interacción de fuerzas, primero creamos un grid de partículas y estudiamos su trayectoria interpolando los movimientos cercanos a estas. Las zonas con mayor movimiento en una misma dirección se marcan como más fuertes, y en función de dichas fuerzas se detectan comportamientos anómalos.

- Codificar el vídeo por medio de un modelo entrenado con vídeos no anómalos. Puesto que dicho modelo solo conoce los vídeos normales, al intentar reconstruir una anomalía se obtendrá un resultado más irregular o con una mayor tasa de error, lo que nos marcará que se está produciendo una anomalía. Un ejemplo de esto se puede ver en [5] que usa un diccionario de códigos binarios y un histograma de dichos códigos como representación.
- Usar modelos *end-to-end* de *deep learning*, como en [6], donde se entrenan 2 redes adversarias: un generador y un discriminador, y usa el discriminador para clasificar cada frame como normal o anómalo.

En nuestro caso queremos centrarnos en un modelo que extraiga un conjunto de características de forma analítica, aprovechando el conocimiento y la intuición que ya tenemos sobre el tipo de anomalías esperadas. Dentro de los trabajos que siguen este enfoque el que parece aportar más información y mejores resultados es el que trataremos a continuación.

3.2.1. Modelo basado en descriptores visuales locales de nivel medio

El modelo sobre el que vamos a basar nuestra propuesta es el expuesto en [7]. Este artículo propone un proceso analítico de extracción de características, que se realiza a partir de una representación espacio-temporal de la multitud. Esta representación está formada por un conjunto de puntos de interés extraídos por un algoritmo de detección como FAST [8].

Los puntos son rastreados a lo largo del tiempo, guardando las trayectorias que siguen, representando así la información temporal del modelo. Para hacer esto se usa un rastreador de características dispersas como ROLF [9] o el rastreador de Kanade–Lucas–Tomasi [10].

Para representar la información espacial, cada punto está relacionado con sus vecinos por medio de una triangulación de Delaunay. Dicha triangulación es una red de triángulos conexa y convexa en donde la circunsferencia circunscrita de cada triángulo no contiene ningún vértice de otro triángulo. También podría usarse un algoritmo de los K vecinos más cercanos, pero el artículo sostiene que el grafo aporta una mejor representación de la información local y espacial.

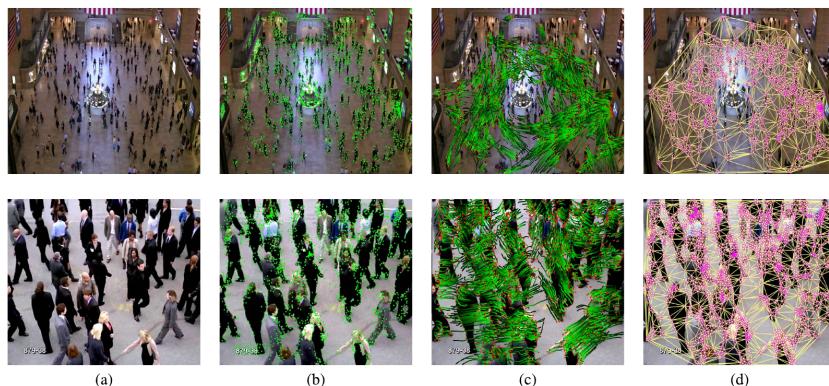


Figura 3.4: Ejemplo extraido de [7]. (a) Fotograma de ejemplo. (b) Detección de puntos de interés con FAST. (c) Trayectorias de los puntos a lo largo del tiempo. (d) Triangulación de Delaunay.

Una vez representada la información de la multitud, se calcula una serie de descriptores por cada punto detectado. Los descriptores son los siguientes:

- **Velocidad** de cada punto rastreado.

- **Variación del flujo dirección:** cuanto más recta sea la trayectoria, menor será este valor.
- **Estabilidad** de la estructura de la multitud: una multitud que mantenga una misma forma durante mucho tiempo será estable, pero una que se mueve de forma errática no.
- **Cohesión** (*Collectiveness*): grado en el que los individuos se mueven de manera conjunta y en una misma dirección.
- **Conflictos:** medición del nivel de interacción entre personas cercanas.
- **Densidad local:** número aproximado de personas en una determinada zona, calculado en función del número de puntos localizados en esa zona.
- **Uniformidad:** indicador de si los puntos rastreados se distribuyen de manera uniforme o se concentran en ciertas zonas formando subgrupos.

Una vez hechos todos los cálculos se generan histogramas para cada descriptor, se juntan para formar un vector de características y se entrena un SVM de clasificación binaria.

En el propio artículo podemos ver los resultados de este modelo en los 2 conjuntos de datos que vamos a usar, obteniendo una puntuación de 88%, usando la métrica del área por debajo de la curva (AUC), en el conjunto de Violent Flows, y un 98.6% en el de UMN. Estos resultados son especialmente buenos, sobre todo si nos fijamos en las comparaciones que se exponen en el mismo artículo con respecto a trabajos similares.

Por otro lado, también menciona la capacidad de este sistema de funcionar a tiempo real, aunque solo indica que siendo implementado en C++ funciona a 5 fotogramas por segundo “*sin estar optimizado*”. Esta afirmación es demasiado vaga como para ser tomada en cuenta, ya que la velocidad de este algoritmo depende en gran medida del número de puntos captados, algo que puede variar en gran medida en función del vídeo o de los parámetros elegidos. Más adelante expandiremos más sobre este tema.

3.3. Modelo propuesto

En esta sección hablaremos del proceso de implementar un sistema lo más cercano posible al descrito en el apartado anterior, así como los cambios

que se han propuesto para mejorarlo. La mayoría de lo que se mencione en este apartado será común tanto al modelo descrito en [7], como a nuestro modelo, ya que estamos partiendo de este, pero se señalarán los casos en los que se diverja del diseño original.

El mayor obstáculo que se ha encontrado para intentar replicar este modelo es que aunque está lo suficientemente bien explicado como para entender su funcionamiento, a la hora de entrar en detalles de implementación, e incluso de evaluación sobre los conjuntos de datos, hay apartados en los que no disponemos de suficiente información como para replicar fielmente lo expuesto en el artículo. iremos mencionando estos casos conforme vayamos avanzando.

3.3.1. Representación de la multitud

Como comentábamos en la sección anterior, el primer paso para detectar una anomalía es representar la información espacio-temporal de la multitud. Para esto primero detectamos un conjunto de puntos de interés mediante el algoritmo FAST (Features from Accelerated Segment Test) [8]. El fin de este algoritmo es buscar esquinas en la imagen, y dada la forma y apariencia de una persona, se detectará un gran número de estos puntos en cada persona. Para distinguir a los puntos que corresponden con una persona de los del fondo de la imagen usaremos un filtro basado en su velocidad de movimiento. Puesto que la velocidad de cada punto es uno de los descriptores que vamos a usar, podemos usar este valor para eliminar todos los puntos que se encuentren por debajo de un umbral. Este filtro suele funcionar bien en escenas en las que la cámara está fija, pero como veremos más adelante este no siempre es el caso, e incluso hay factores que pueden complicar su buen funcionamiento aunque se cumpla esta condición.

Una vez tenemos un conjunto de puntos localizados usaremos el rastreador de Kanade–Lucas–Tomasi, que nos puede indicar la posición siguiente de un punto a partir de dos fotogramas consecutivos y su localización original. Para una mayor fiabilidad en el rastreo usaremos un esquema de verificación hacia delante y hacia atrás, es decir, después de obtener la posición final de un punto la verificaremos realizando el rastreo a la inversa, si obtenemos de nuevo la posición original tomaremos el resultado como bueno, y si no consideramos que hemos perdido la trayectoria y eliminaremos el punto.

La forma de gestionar las trayectorias será el primer aspecto en el que

nos separaremos de [7]. En este artículo se menciona una alternancia entre la detección de nuevos puntos y el rastreo de los ya existentes en función de la fiabilidad de la trayectoria, sin embargo demasiados aspectos no se detallan, como la manera de incorporar estos nuevos puntos en el sistema sin sobrecargarlo de posiciones muy cercanas entre si, o el criterio para detectar a personas que hayan podido entrar en escena.

Es por eso que en su lugar se ha decidido seguir un esquema más sencillo en el que si usamos trayectorias de longitud L , primero llenamos la trayectoria del punto durante esos L fotogramas y luego calculamos los descriptores de ese punto durante otros L fotogramas. De esta forma, detectariamos nuevos puntos, como mucho, cada $2L$ fotogramas. Puesto que este método nos obligaría a renunciar a la información de la mitad de los fotogramas usaremos 2 conjuntos de trayectorias, una que usaremos para calcular los descriptores y otra en la que iremos construyendo la trayectoria de los nuevos puntos.

Cabe destacar que este segundo conjunto de trayectorias podría desecharse en una implementación orientada a un uso práctico en la que no fuese importante obtener toda la información de los vídeos, pero los conjuntos de datos con los que trabajamos son bastante pequeños y las precisiones lo bastante altas como para que cada fotograma cuente.

Por otro lado, como decíamos en el apartado anterior, usaremos una triangulación de Delaunay para relacionar espacialmente los puntos de cada fotograma. OpenCV ya nos proporciona una clase que llevará a cabo este proceso, llamada SubDiv2D. Una vez creado el grafo en los siguientes apartados haremos uso del concepto de clique de primer orden para relacionar unos puntos con otros. Definimos el clique del punto V_i como todos los puntos V_j del grafo tales que existe una arista que une V_i y V_j . En [7] se habla también de los cliques de orden mayor que uno. Estos cliques se definen de manera recursiva de forma que un clique de orden k se forma añadiendo al clique de orden $k-1$ los puntos conectados con cualquier miembro este. Puesto que en ningún momento se menciona que estos aporten nada de valor al cálculo, y además añadirían un gran peso computacional, ignoraremos estos cliques de orden superior y usaremos solo cliques de primer orden.

Resumiendo todo el proceso de este apartado nos quedamos con 3 pasos principales. Primero captar puntos de interés con FAST, que más adelante serán filtrados en función de su velocidad. Segundo, representar la evolución

temporal de la multitud por medio de las trayectorias de cada punto a lo largo de un determinado número de fotogramas. Tercero, representar las relaciones espaciales de los puntos en cada fotograma por medio de los cliques de primer orden presentes en una triangulación de Delaunay que contenga a todos los puntos.

3.3.2. Descriptores visuales

En este apartado explicaremos en mayor profundidad los descriptores que usaremos para representar el estado de la multitud. Estos se calculan usando a partir de las representaciones explicadas en el apartado anterior. Existen 3 parámetros que determinarán que momento de la trayectoria usamos para realizar ciertos cálculos. El primero es el tamaño total de la trayectoria, que denotaremos como L . Puesto que estas trayectorias miden un periodo de tiempo relativamente amplio también usaremos otros dos parámetros, τ_1 y τ_2 , que marcarán momentos intermedios de la trayectoria que usaremos como referencia a la hora de calcular algunos valores como la velocidad.

- **Velocidad:** en [7] se calcula este valor como la distancia euclídea de la posición actual de un punto y su posición τ_1 fotogramas antes. Puesto que de la cámara no suele ser perpendicular al plano en el que se mueve la multitud, los desplazamientos en el eje x e y tienen ordenes de magnitud distintos. Para solucionar esto [7] crea un mapa de perspectiva interpolando la altura de una persona de referencia en dos extremos de la imagen. Esta idea está insuficientemente detallada como para poder ser replicada con fidelidad, y además solo es aplicable a conjuntos de datos en los que los videos comparten una misma escena. Es por esto que en su lugar se ha decidido dividir este descriptor en 2 que se traten por separado, uno que mida la velocidad en el eje x y otro que la mida en el eje y . En cuanto al filtro que mencionábamos en el apartado anterior, una vez calculada la distancia que recorre un punto en el eje x e y , si la suma de ambos valores (distancia Manhattan) es menor que un valor concreto β , dicho punto es eliminado.
- **Variación del flujo de dirección:** este descriptor busca distinguir entre movimientos suaves y caóticos. Su cálculo será similar al de [7], con la única diferencia que nosotros solo usamos trayectorias de tamaño constante L , lo que nos permite ahorrarnos ciertos cálculos superfluos. Para este cálculo primero seleccionamos F fotogramas separados a una distancia de τ_2 entre si. Dado que las trayectorias tienen un tamaño fijo

de L fotogramas, $F = \lfloor L/\tau_2 \rfloor$. Una vez dividida la trayectoria, calculamos la dirección de cada fotograma seleccionado al siguiente y calculamos la diferencia con la dirección anterior. Este cálculo se puede expresar de la siguiente forma, siendo V_i^k el punto i en el instante de tiempo k , d_θ la diferencia angular (que definiremos a continuación), y $S_i^{k-f\tau_2}$ la posición que tenía el punto i en el instante $k - f\tau_2$:

$$D^{var}(V_i^k) = \sum_{f=0}^{F-2} d_\theta(S_i^{k-f\tau_2}, S_i^{k-(f+1)\tau_2}) \quad (3.1)$$

Definimos d_θ como $\Delta\theta(|\theta(a) - \theta(b)|)$, donde $\theta(a)$ es el ángulo que el vector a hace con el eje x . $\Delta\theta(\alpha)$ sirve para corregir la dirección de α y se define como α cuando este es menor o igual que π y $2\pi - \alpha$ cuando es mayor.

- **Estabilidad:** este descriptor intenta estudiar la estructura topológica de la triangulación de Delaunay a lo largo del tiempo. Para esto comparamos la forma de este en el instante actual, con la que tenía hace τ_2 fotogramas. En [7] se intenta tener en cuenta tanto la distribución de los cliques como los puntos que son eliminados e insertados en el gráfico, sin embargo esto supone una gran cantidad de elementos a guardar en memoria (p.e. el grafo de cada fotograma) y de cálculos (volver a calcular los cliques de fotogramas pasados). En nuestro caso nos quedaremos solo con el elemento principal de este descriptor, la deformación de los triángulos, y asumiremos que los cliques están estructurados de la misma forma en ambos fotogramas. De esta forma, por cada clique de grado 1 podremos sacar un conjunto de triángulos $r_{i\alpha}$. Por cada triángulo que tengamos en un clique calcularemos su diferencia con el triángulo formado por la posición de esos tres mismos puntos τ_2 fotogramas antes. La diferencia de ambos triángulos se define de la siguiente manera:

$$g(r_{i\alpha}, r_{i\beta}) = |a_{i\alpha} - a_{i\beta}| * |c_{i\alpha} - c_{i\beta}| \quad (3.2)$$

Donde $a_{i\alpha}$ es el área del triángulo y $c_{i\alpha}$ es el *cross-ratio*, que se calcula con los dos vértices del triángulo que no son i ($V_\alpha, V_{\alpha'}$), y las proyecciones de los puntos medios de los lados que van a i ($p'_{i\alpha}, p'_{i\alpha'}$). Estos 4 puntos se ordenan en una misma dirección, formando un vector (x_1, x_2, x_3, x_4) con el que se calcula la *cross-ratio* de la siguiente manera:

$$f_{cr}(x_1, x_2, x_3, x_4) = \frac{|x_1 - x_3| * |x_2 - x_4|}{|x_1 - x_4| * |x_2 - x_3|} \quad (3.3)$$

Sumando el resultado de todos los triangulos de un clique obtenemos su estabilidad.

- **Cohesión:** este valor busca estudiar el grado en el que los individuos que forman la multitud se mueven en una misma dirección. Se calcula usando la siguiente fórmula:

$$D^{coh}(V_i^k) = \frac{1}{|C_n V_i^k|} \sum_{V_j^k \in C_n(V_i^k)} d_\theta(\overrightarrow{V_i^{k-\tau_1} V_i^k}, \overrightarrow{V_j^{k-\tau_1} V_j^k}) \quad (3.4)$$

Donde $C_n V_i^k$ es el conjunto de puntos pertenecientes al clique de orden 1 de V_i en el instante k . Esta fórmula es igual que la que se usa en [7] salvo por el uso de la función d_θ , que en su caso se compara primero con un valor T_1 y si es menor que este se sustituye por un 0. Puesto que en ningún momento se menciona qué es T_1 , por qué debería usarse o que valor debería tomar se ha decidido prescindir de él. Contrariamente a lo que podríamos pensar, esta fórmula tomará valores más bajos conforme más cohesionada esté la multitud, y tomará valores más altos cuando las trayectorias de los puntos de un mismo clique sean muy distintas.

- **Conflicto:** el objetivo de este descriptor es caracterizar la interacción entre los miembros de la multitud. Se calcula de manera muy similar a la cohesión, y de hecho podemos implementar ambos valores dentro de una misma función para evitar repetir cálculos innecesarios. Su fórmula es la siguiente:

$$D^{conf}(V_i^k) = \frac{1}{|C_n V_i^k|} \sum_{V_j^k \in C_n(V_i^k)} \frac{d_\theta(\overrightarrow{V_i^{k-\tau_1} V_i^k}, \overrightarrow{V_j^{k-\tau_1} V_j^k})}{\|V_i^k V_j^k\|_2} \quad (3.5)$$

Como se puede ver, la única diferencia es que en este caso dividimos por la distancia de V_i^k a V_j^k , lo que, como es de esperar, nos dará valores de conflicto más altos cuando los puntos estén cercanos entre si.

- **Densidad:** este valor describe la cercanía de los puntos de un mismo clique. se calcula de la siguiente forma:

$$D^{dens}(V_i^k) = \frac{1}{\sqrt{2\pi}\sigma} \sum_{V_j^k \in C_n(V_i^k)} \exp - \frac{\|V_i^k V_j^k\|_2}{2\sigma^2} \quad (3.6)$$

Donde σ es el ancho de banda del kernel Gaussiano que determina el efecto de cada vecino en la densidad. En ciertos casos este valor debería ajustarse mediante un mapa de perspectiva, pero puesto que en la

mayoria de las escenas en las que trabajaremos no hay una gran distorsión de la perspectiva usaremos un valor constante que ajustaremos empíricamente.

- **Uniformidad:** este descriptor refleja la homogeneidad de la distribución de los distintos subgrupos dentro de una misma multitud. El primer paso para calcular este valor es usar un algoritmo de *clustering* para agrupar los puntos del grafo en grupos. En [7] no se menciona ningún algoritmo concreto, y solo se recomienda que se use un algoritmo basado en la distancia que no necesite conocer el número de grupos de antemano. Dentro de este tipo de algoritmos uno de los más usados y que mejor funciona es DBSCAN [11]. Una vez tenemos un conjunto de p grupos, $N = N_1, N_2, \dots, N_p$ calculamos la uniformidad de un grupo como:

$$D^{unif}(N_i) = \frac{A(N_i, N_i)}{A(N, N)} - \left(\frac{A(N_i, N)}{A(N, N)} \right)^2 \quad (3.7)$$

$$A(N_i, N_i) = \sum_{p \in N_i} \sum_{q \in C_1(p), q \in N_i} \frac{1}{\|pq\|_2} \quad (3.8)$$

$$A(N_i, N) = \sum_{p \in N_i} \sum_{q \in C_1(p), q \notin N_i} \frac{1}{\|pq\|_2} \quad (3.9)$$

$$A(N, N) = \sum_{i \in N} \sum_{p \in N_i} \sum_{q \in C_1(p)} \frac{1}{\|pq\|_2} \quad (3.10)$$

En resumen, la uniformidad de un grupo es alta si la distancia de los puntos de una misma clase es pequeña y la distancia entre puntos de distintas clases es grande. Por último, aunque este descriptor se calcula por grupos, para mantener la homogeneidad con el resto de descriptores le asignaremos a cada punto el valor de su grupo correspondiente. De esta forma también podremos tener en cuenta, de manera indirecta, el número de puntos de cada grupo cuando juntemos estos valores en un histograma.

3.3.3. Vector de características y clasificador

Una vez hemos calculado los descriptores para cada punto rastreado solo nos queda procesar esta información mediante técnicas de aprendizaje automático para obtener una predicción. Puesto que el número de puntos puede variar en cada fotograma, construiremos un histograma para cada descriptor en cada frame. Un histograma es una representación de la distribución numérica de un conjunto de valores dentro de un rango. Se construye dividiendo

en n partes iguales el rango indicado, y contando el número de valores que entra en cada partición. Con esta representación podemos estimar si la distribución de valores es normal o si hay una cantidad anormal de valores altos o bajos. Para el rango de los histogramas usaremos 0 como valor mínimo, ya que es el valor mínimo de todos los descriptores, y como valor máximo usaremos el percentil 95 de los datos que usemos para entrenar. Usamos un percentil en lugar del máximo absoluto porque valores anórmalmente altos podrían afectar demasiado al rango provocando que la mayoría de los valores se concentren en el primer intervalo, perdiendo por tanto mucha precisión. En [7] se menciona usar $n = 16$, pero en nuestro caso se ha encontrado que 32 suele funcionar mejor. Puesto que tenemos 8 descriptores (recordemos que la velocidad se divide en 2) usariamos vectores de unos 256 elementos.

En dicho artículo este vector se usaría directamente con un SVM para entrenar el modelo o para predecir la clase de un fotograma o un vídeo, pero en nuestro caso proponemos 2 alternativas para procesar el vector en uno más pequeño y más fácil de clasificar. Estas propuestas consisten en usar uno de los 2 métodos de reducción de dimensionalidad más usados, el PCA y los *autoencoders*, para transformar el vector de histogramas, en una representación más compacta y con valores más relevantes. De esta forma, no solo podríamos ahorrar espacio de almacenamiento al usar valores más pequeños, si no que podríamos aumentar el número de particiones de los histogramas, reduciendo el sesgo que produce este tipo de representación, y obtener más información sin que el mayor número de variables perjudique a la precisión de nuestro modelo. A continuación pasaremos a discutir ambas posibilidades en mayor detalle.

Análisis de Componentes Principales (PCA)

El análisis de componentes principales es una transformación lineal que traslada los datos a un nuevo sistema de coordenadas. En este nuevo sistema los ejes están ordenados de tal manera que la mayor varianza se produce en la primera coordenada, y la menor en la última. Este orden en los ejes es lo que nos permite usar esta técnica para reducir la dimensionalidad de nuestros datos, usando solamente los primeros ejes que se obtengan. Generalmente es posible mantener más del 95% de la varianza de los datos con una dimensionalidad mucho menor.

Para usar esta técnica primero hay que asumir una serie de condiciones que no siempre se dan. Una de estas condiciones es que los valores sean inde-

pendientes entre si, lo que no es cierto en nuestro caso, sin embargo, creemos que merece la pena comprobar empíricamente su efecto, dado que en [7] ya se habla de aplicar PCA y se muestra una mejora en el modelo, aunque esta no se exponga como parte de los resultados finales. Además es una técnica de fácil implementación y en ciertos casos se ha conseguido llegar a buenos resultados a pesar de no cumplir estas condiciones.

Autoencoder

Puesto que la reducción de dimensionalidad podría ayudar a la clasificación y el PCA puede no ser la técnica más adecuada para nuestro problema, proponemos el uso de otra gran técnica de reducción de dimensionalidad, los *autoencoders*. Ya hemos hablado de ellos en el apartado de fundamentación teórica, pero recordemos que un *autoencoder* se trata de una red neuronal que se divide en dos partes, un codificador y un decodificador, y que se entrena de manera no supervisada. El codificador primero reduce la entrada a un tamaño indicado por su capa de código, mientras que el decodificador intenta expandir la información de dicha capa para reconstruir la entrada. Una vez entrenada la red se extrae el codificador y se usa para reducir la dimensionalidad de los nuevos datos en un vector de características más pequeño pero elaborado.

La gran ventaja de estas redes es su adaptabilidad, ya que no necesitan cumplir ninguna serie de condiciones especiales. Además, puesto que se trata de una red neuronal, podemos ajustar su entrenamiento para que tenga en cuenta la información de clase y favorecer las posibles codificaciones que sean más fáciles de clasificar. Para hacer esto haremos uso del entrenamiento multi-tarea, donde para cada entrada tendremos 2 salidas, cada una con un valor de pérdida distinto. La primera salida será la del decodificador, y su pérdida será el valor del error cuadrático al comparar la entrada con la salida obtenida. La segunda salida será una clasificación, al igual que si usaramos una red neuronal normal para clasificar el vector de características como normal o anómalo. Es importante señalar que usamos esta segunda rama de la red para ayudar al codificador, ya que en este caso una SVM es más efectiva a la hora de clasificar.

En cuanto a los inconvenientes de esta propuesta nos encontramos con la mayor complejidad que supone diseñar una red neuronal, especialmente comparado con un PCA, y que entrenar este tipo de codificadores suele necesitar más datos que su alternativa, algo de lo que hablaremos en el apartado

de resultados.

Resultados experimentales

4

4.1. Metodología usada

4.1.1. UMN Crowd Dataset

4.1.2. Violent Flows

4.2. Comparación de resultados

4.2.1. UMN Crowd Dataset

4.2.2. Violent Flows

5 Conclusión

Conclusión

Bibliografía

- [1] UMN Crowd Dataset. http://mha.cs.umn.edu/proj_events.shtml#crowd/.
- [2] Violent Flows Dataset. <https://www.openu.ac.il/home/hassner/data/violentflows/>.
- [3] Duan-Yu Chen y Po-Chung Huang. “Motion-based unusual event detection in human crowds”. En: *J. Visual Communication and Image Representation* 22 (feb. de 2011), págs. 178-186. doi: [10.1016/j.jvcir.2010.12.004](https://doi.org/10.1016/j.jvcir.2010.12.004).
- [4] R. Mehran, A. Oyama y M. Shah. “Abnormal crowd behavior detection using social force model”. En: (2009), págs. 935-942.
- [5] Mahdyar Ravanbakhsh y col. “Plug-and-Play CNN for Crowd Motion Analysis: An Application in Abnormal Event Detection”. En: (oct. de 2016).
- [6] Mahdyar Ravanbakhsh y col. “Training Adversarial Discriminators for Cross-channel Abnormal Event Detection in Crowds”. En: (jun. de 2017).
- [7] Hajar Fradi, Bertrand Luvison y Quoc-Cuong Pham. “Crowd Behavior Analysis Using Local Mid-Level Visual Descriptors”. En: *IEEE Transactions on Circuits and Systems for Video Technology* PP (oct. de 2016), págs. 1-1. doi: [10.1109/TCSVT.2016.2615443](https://doi.org/10.1109/TCSVT.2016.2615443).
- [8] Edward Rosten, Reid Porter y Tom Drummond. “Faster and Better: A Machine Learning Approach to Corner Detection”. En: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32.1 (2010), págs. 105-119. doi: [10.1109/TPAMI.2008.275](https://doi.org/10.1109/TPAMI.2008.275).
- [9] Tobias Senst, Volker Eiselein y Thomas Sikora. “Robust Local Optical Flow for Feature Tracking”. En: *IEEE Transactions on Circuits and Systems for Video Technology* 22 (sep. de 2012). doi: [10.1109/TCSVT.2012.2202070](https://doi.org/10.1109/TCSVT.2012.2202070).
- [10] Carlo Tomasi y Takeo Kanade. *Detection and Tracking of Point Features*. Inf. téc. International Journal of Computer Vision, 1991.

- [11] Michael Hahsler, Matthew Piekenbrock y Derek Doran. “dbscan: Fast Density-Based Clustering with R”. En: *Journal of Statistical Software* 91.1 (2019), págs. 1-30. doi: [10.18637/jss.v091.i01](https://doi.org/10.18637/jss.v091.i01).